IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF WISCONSIN

WISCONSIN ALUMNI RESEARCH FOUNDATION	)	
Plaintiff,	)	
v.	)	**Case No. 08-C-78-C**
INTEL CORPORATION	)	
Defendant.	)	

PLAINTIFF'S MOTION FOR CONSTRUCTION OF CLAIMS AND OPENING BRIEF IN SUPPORT OF MOTION

TABLE OF CONTENTS

TABLE OF AUTHORITIES

CASES Page

MOTION

Plaintiff Wisconsin Alumni Research Foundation ("WARF") hereby moves for construction of the claim terms "prediction" and "in fact executed" found in claim 1 of U.S. Patent No. 5,781,752, and for a hearing on the construction of these claim terms. As discussed in greater detail in the accompanying brief, the Court's construction of these claim terms is potentially relevant to the issue of validity and will assist the factfinder in understanding the scope of the claims at issue.

In addition, WARF hereby moves for the Court to hold a hearing to assist its determination of the proper construction of these claim terms. The patent at issue in this case is in the complex field of computer microarchitecture. In addition to a basic familiarity with the unique terminology used in the computer sciences, understanding the claimed invention and the disputed issues will require an appreciation of a number of intricate concepts relating to the processing of instructions by a microprocessor (e.g., "out of order" and "speculative" execution of instructions; data dependences between instructions; memory hierarchies). Because of the complexity of the technology at issue, WARF also requests the opportunity to present a tutorial for the Court through its expert, Professor William Dally, at such hearing, in addition to addressing the construction of the claim terms themselves.

OPENING BRIEF IN SUPPORT OF MOTION

INTRODUCTION

Our story begins in the early 1990s in the computer sciences building on the campus of the University of Wisconsin-Madison, where a professor (Gurindar Sohi) and three of his graduate students (Andreas Moshovos, Scott Breach, and Terani Vijaykumar) had been working on the design of a Multiscalar computer processor. After years of experimenting with and refining the design, in the fall of 1995, they finally came up with a method for improving processor performance by selective data speculation of LOAD instructions. Upon conceiving the invention, they immediately raced off to test this novel idea in their simulator for the Multiscalar processor and – from even the earliest, most rough results spit out by the simulator – they all knew that they were onto something important.

Thus was born the invention claimed in U.S. Patent No. 5,781,752, issued on July 14, 1998 and assigned to plaintiff WARF.[1] Specifically, the '752 patent discloses and claims a data speculation decision circuit to facilitate the advanced execution of instructions before other instructions on which they may be data dependent, more specifically, the selective data speculation of LOAD instructions. This pioneering invention significantly enhanced the opportunities for instruction-level parallelism and thereby improved execution efficiency and speed; as a result, the invention claimed in the '752 patent has been widely recognized as a significant advance in computer architecture both by researchers in the field and those in industry.

[1] A copy of the '752 patent is attached as Exhibit 1 to the Declaration of Michelle M. Umberger in Support of Plaintiff's Motion for Construction of Claims and Opening Brief in Support of Motion, filed herewith (hereinafter "Umberger Decl."). A bound copy of the certified file history of the '752 patent has been filed separately with the Court.

In this action, WARF alleges that Intel's sale of processors based on or derived from the Intel Core™ Microarchitecture and the enhanced Intel Core™ Microarchitecture infringe claims 1 and 2 of the '752 patent. WARF believes that the following claim terms or phrases would benefit from the Court's construction[2]:

- "prediction"
- "in fact executed"

While Intel's proposed constructions of these claim terms take the terms out of context, WARF has focused on the claim language and the specification, advancing proposed constructions intended to provide a uniform and coherent treatment of the claims consistent with the understanding of those of ordinary skill in the art.

LEGAL STANDARDS

"It is a bedrock principle of patent law that the claims of a patent define the invention to which the patentee is entitled the right to exclude." *Phillips v. AWH Corp.*, 415 F.3d 1303, 1312 (Fed. Cir. 2005) (internal quotation marks omitted); *see also John Mezzalingua Assocs. v. Arris Int'l, Inc.*, 298 F. Supp. 2d 813, 817 (W.D. Wis. 2003) (Crabb, C.J.) ("[T]he language of the claim defines the scope of the protected invention.") (internal quotation marks omitted). When

[2] In the meet and confer process, WARF initially proposed only a single phrase for construction, namely, "a prediction associated with the particular data consuming instruction and based on the mis-speculation indication." Intel counter-proposed numerous other claim terms for construction, and WARF accordingly proposed alternate constructions for those terms as part of the meet and confer process. The parties came to agreement as to the proper constructions of several claim terms, namely "data dependence between instructions," "data speculation," "speculation," and "range." These agreed constructions are included in the joint chart attached as Umberger Decl. Ex. 2. It is WARF's position that only the additional terms "in fact executed" and "prediction" will benefit from the Court's construction. To the extent that Intel proposes other terms in its motion, WARF will offer its objections (on both procedural and substantive grounds) in its opposition brief.

construing the claims of a patent, "the starting point is the so-called intrinsic evidence: the claims themselves, the patent specification and the prosecution history." *Gen. Elec. Co. v. Sonosite, Inc.*, No. 07-CV-00273-bbc, 2008 U.S. Dist. LEXIS 33223, at *5 (W.D. Wis. Jan. 8, 2008) (citing *Teleflex, Inc. v. Ficosa N. Am. Corp.*, 299 F.3d 1313, 1325 (Fed. Cir. 2002)). Courts first look to the claim terms themselves, which are to receive their "ordinary and customary meaning," that is, "the meaning that a person of ordinary skill in the art would have understood the claim term to have as of the filing date of the patent application." *Id.* at *5-6 (citing *Phillips*, 415 F.3d at 1313).

While "the claims themselves provide substantial guidance as to the meaning of particular claim terms[,]" they "do not stand alone[,]" but rather "are part of a fully integrated written instrument." *Phillips*, 415 F.3d at 1314-15 (internal quotation marks omitted). Thus, a person of ordinary skill in the art "is deemed to read the claim term not only in the context of the particular claim in which the disputed term appears, but in the context of the entire patent, including the specification." *Phillips*, 415 F.3d at 1313. The claims must be "construed to preserve the patent's internal coherence." *Applied Med. Res. Corp. v. U.S. Surgical Corp.*, 448 F.3d 1324, 1333 n.3 (Fed. Cir. 2006) (internal quotation marks omitted).

"The specification serves an important role in arriving at the correct claim construction because it is there that the patentee provides a written description of the invention that allows a person of ordinary skill in the art to make and use the invention." *Gen. Elec. Co.*, 2008 U.S. Dist. LEXIS 33223, at *6. As the Federal Circuit has often repeated, "the specification is always highly relevant to the claim construction analysis. Usually, it is dispositive; it is the single best guide to the meaning of a disputed term." *Chamberlain Group v. Lear Corp.*, 516 F.3d 1331, 1335 (Fed. Cir. 2008) (internal quotation marks omitted).

Given the importance placed on the specification, a court cannot "interpret claim terms in a way that excludes embodiments" included therein. *Oatey Co. v. IPS Corp.*, 514 F.3d 1271, 1276 (Fed. Cir. 2008) (citations omitted) (concluding that the district court improperly excluded from the scope of the claim an embodiment disclosed in a figure in the specification). But it is also true that a claim is not limited only to the embodiments disclosed in the specification. *Phillips*, 415 F.3d at 1323 ("[A]lthough the specification often describes very specific embodiments of the invention, we have repeatedly warned against confining the claims to those embodiments."). Instead, "[t]o avoid importing limitations from the specification into the claims, it is important to keep in mind that the purposes of the specification are to teach and enable those of skill in the art to make and use the invention and to provide a best mode for doing so." *Id.*

After reviewing the claim language and patent specification, a court may also consider the prosecution history, which "provides evidence of how the PTO and the inventor understood the patent." *Phillips*, 415 F.3d at 1317; *see also Gen. Elec. Co.*, 2008 U.S. Dist. LEXIS 33223, at *7. Finally, "a court may consult extrinsic evidence, such as dictionaries, treatises and expert testimony." *Gen. Elec. Co.*, 2008 U.S. Dist. LEXIS 33223, at *8; *see also Phillips*, 415 F.3d at 1317-18. While of less weight than intrinsic evidence, "extrinsic evidence in the form of expert testimony can be useful to a court for a variety of purposes, such as to provide background on the technology at issue, to explain how an invention works, to ensure that the court's understanding of the technical aspects of the patent is consistent with that of a person of skill in the art, or to establish that a particular term in the patent or the prior art has a particular meaning in the pertinent field." *Phillips*, 415 F.3d at 1318.

BACKGROUND OF THE TECHNOLOGY

As noted in WARF's motion, given the complexity of the technology at issue in this case, WARF requests the opportunity to provide the Court with a tutorial of the technology claimed in the '752 patent at the time tentatively reserved for a *Markman* hearing (regardless of whether the Court decides to hold a hearing to construe the terms themselves). WARF believes it may be beneficial for the Court to hear the testimony of its expert, Professor William Dally, regarding the technology, and to have an opportunity directly to ask him questions. A summary of the background of the technology is provided below; a more detailed discussion of the technology is set forth in the Declaration of William J. Dally in Support of Plaintiff's Motion for Construction of Claims and Opening Brief in Support of Motion (hereinafter "Dally Decl."), filed herewith, at ¶¶ 7-48.

I. Computer Architecture Basics[3]

A computer system can be subdivided into two functional entities: hardware and software. The software consists of operating system software and applications software, such as a Web browser or a word processor. The hardware portion includes a processor, memory and peripherals. The software consists of sequences of instructions that define a computation. The processor performs that computation by executing the associated sequence of instructions. The memory provides storage for the sequences of instructions and the data associated with the software.

[3] *See* Dally Decl. ¶ 7.

II. Execution Of Instructions In A Processor[4]

At a high level, the role of a processor can be described as repeatedly fetching instructions and data, executing the instructions to modify that data, and saving the results of those executions. *See* Umberger Decl. Ex. 1, Col. 1, ll. 28-30. To perform these tasks, a processor consists of a set of registers that temporarily hold instructions and data that the processor is working upon and execution units to execute those instructions. *See* Umberger Decl. Ex. 1, Col. 1, ll. 2-25. To illustrate how instructions are executed, consider the following four instructions that a processor may find in an Instruction Cache, a memory devoted to storing instructions:

```
1:   LOAD Memory[200], R1
2:   STORE R2, Memory[R1]
3:   LOAD Memory[300], R3
4:   ADD R3, #16, R4
```

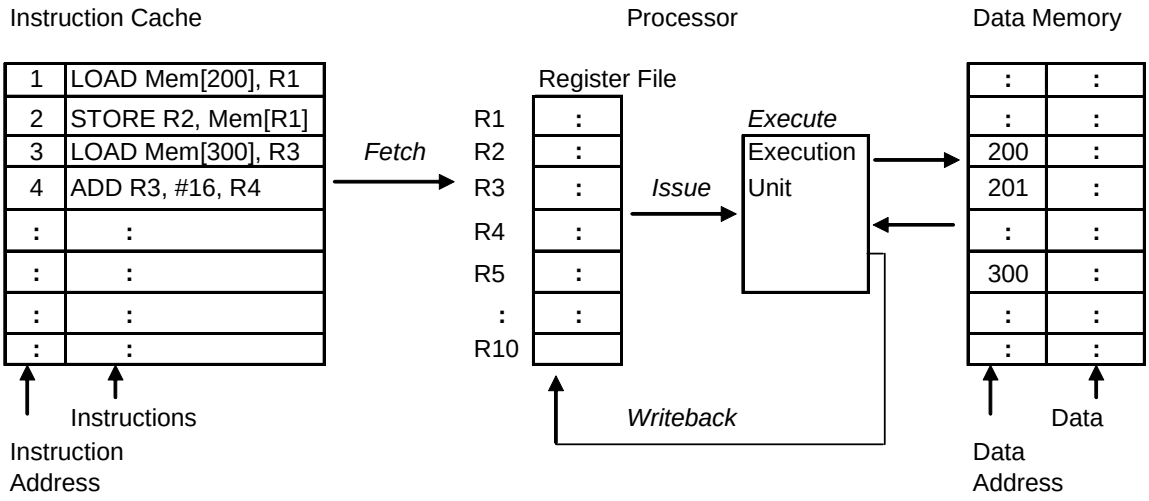

Fig. 1 – Execution of Instructions

A processor begins by fetching the first LOAD instruction (1: LOAD Memory[200], R1) from the Instruction Cache. This instruction directs the processor to retrieve data from the

[4] *See* Dally Decl. ¶¶ 8-9; 11-13.

memory location specified by address 200 and to place that data in register R1. The processor then fetches the next instruction, which is a STORE instruction (2: STORE R2, Memory[R1]). This instruction directs the processor to save the contents of register R2 in the memory location whose address is specified by contents of register R1. The third instruction, which is another LOAD instruction, directs the processor to load the contents of memory at address 300 into register R3. Finally, the ADD instruction directs the processor to add the number 16 to the contents of register R3 and to save the result in register R4. Thus, for each instruction the processor essentially performs three steps in sequence: *fetch* (retrieve the instruction from the Instruction Cache), *issue* (send the instruction and the data required by the instruction to the execution unit), and *execute* (execute the instruction, *i.e.*, operate on the data).

As shown above, a computer's data memory is divided into a number of locations that are referenced by addresses using the LOAD and the STORE instructions. For example, the first LOAD instruction reads from address 200 and the STORE instruction writes to address contained in register R1. Similarly, the instructions are associated with the address at which they are stored. For example, the first LOAD and the second LOAD instructions are associated with addresses 1 and 3 respectively and the STORE instruction is associated with the address 2. Consequently, they are referred to as the *1: LOAD*, *3: LOAD* and *2: STORE* instructions.

III. Instructions May Be Executed Out-Of-Order To Improve Performance[5]

A simple processor may execute the four instructions shown above sequentially (*in order*), *i.e.*, fetch and execute one instruction before fetching the next instruction. Modern high-performance processors execute several instructions in parallel to speed-up overall execution and

[5] *See* Dally Decl. ¶¶ 11-14.

may even execute instructions non-sequentially (*out of order*) to improve performance. *See* Umberger Decl. Ex. 1, Col. 1, ll. 49-58. Even though the instructions are executed non-sequentially (*out of order*) inside a modern processor, to the remainder of the computer system, it still appears that the processor is executing instructions sequentially (*in order*). To provide this illusion, a processor capable of out-of-order execution includes additional hardware: a Reorder Buffer (ROB), a Memory Order Buffer (MOB), additional execution units, and an additional instruction processing step, as shown in Figure 2.

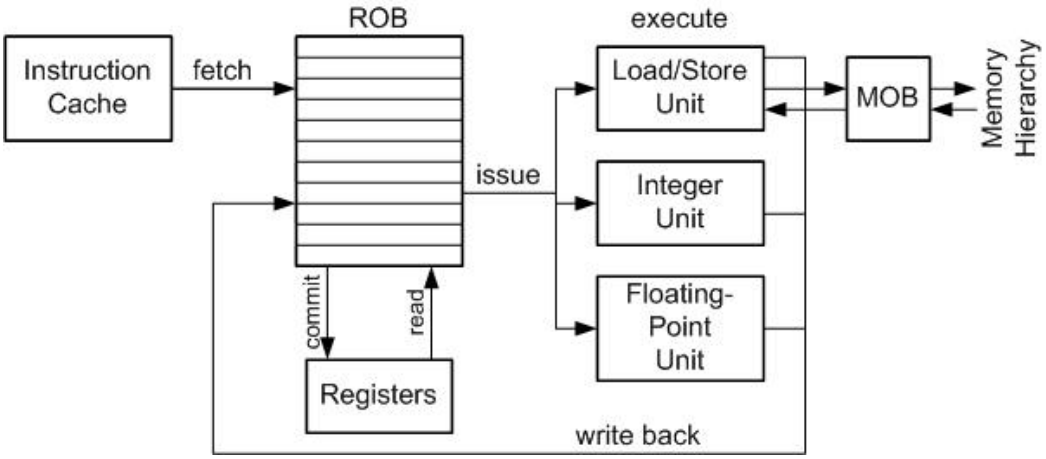

Fig. 2 – Processor Capable of Out of Order Execution

The Reorder Buffer (ROB) holds data and information regarding all the instructions that the processor is working on at any instant of time. The ROB ensures that instructions are handled in a manner that preserves the illusion that these instructions have been executed sequentially. The Memory Order Buffer (MOB) serves the same purpose as the ROB except that the MOB involves only the LOAD and the STORE instructions. Figure 2 depicts the additional execution units: a Load/Store unit that handles only LOAD and STORE instructions, and Integer and Floating point units that handle arithmetic operations. In this processor, the execution unit can operate on at least three instructions in parallel.

To maintain the illusion of sequential execution even though the instructions are executed out of order, the results of those instructions are made available to the remainder of the system only in the sequential order. Thus, to the system its appears that the instruction are being executed in order. To accomplish this, the processor adds a *commit* (or a *retire*) (C) stage after the *fetch* (F), *issue* (I) and *execute* (E) stages. In the commit stage, an instruction transfers its results to locations from where the remainder of the system can access them. In the processor above, in the *commit* stage, a STORE instruction writes to memory whereas non-STORE instructions transfer values from the ROB to the registers. An instruction commits when all other instructions that occur earlier than that instruction have committed. Another attribute of the *commit* stage is that an instruction makes its results permanent only in this stage. Therefore, an instruction can be undone (or discarded) anytime before it *commits* or *retires*. This assists in the speculative execution of instructions, as discussed in detail below.

IV. Data Dependences Must Be Observed In Spite Of Out-Of-Order Execution[6]

To make overlapped or out-of-order execution indistinguishable from sequential execution, the processor must also ensure that the **data dependences** between program instructions are satisfied. A data dependence between two instructions exists if a later instruction uses data from a location to which an earlier instruction may write. *See* Umberger Decl. Ex. 1, Col. 1, l. 67 – Col. 2, l. 2. To illustrate the concept of data dependence, consider the same four instruction sequence:

[6] *See* Dally Decl. ¶¶ 15-23.

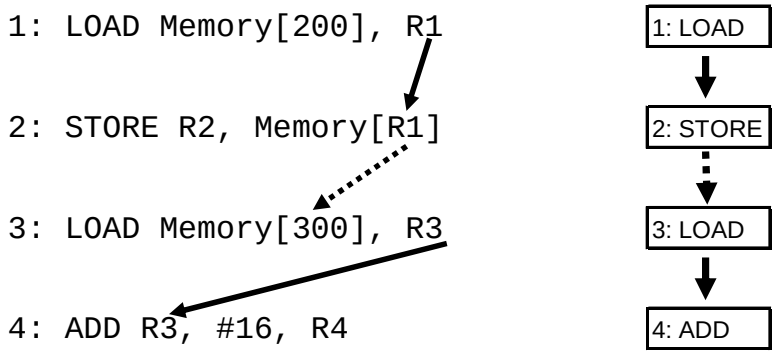

Fig. 3 – Data Dependences Between Instructions

This sequence of instructions contains two *unambiguous data dependences* (denoted by solid arrows in Figure 3):

- The *2: STORE instruction* is data dependent on the *1: LOAD instruction* because the STORE uses data from a location that is modified by the LOAD instruction. The *1: LOAD instruction* updates the contents of register R1 and the *2: STORE instruction* uses the contents of that register to obtain the address to which it will write.
- The *4: ADD instruction* is data dependent upon the *3: LOAD instruction* because the ADD instruction uses the contents of register R3, which is modified by the *3: LOAD instruction*.

Because these data dependences can be determined by simply reviewing the instruction sequence, they are referred to as *unambiguous* data dependences. *See* Umberger Decl. Ex. 1, Col. 2, ll. 5-7. To ensure correct execution of the program, these dependences must be observed, *i.e.*, the *2: STORE* instruction must be executed after the *1: LOAD* instruction and the *4: ADD* instruction must be executed after the *3: LOAD* instruction.

The above sequence also contains one *ambiguous data dependence*, which is denoted by a dotted arrow. Consider the *2: STORE* and the *3: LOAD* instructions from the sequence above. The *2: STORE* instruction writes to address specified by register R1 whereas the LOAD instruction reads from memory address 300. There are two possibilities:

- If the address specified by register R1 and address 300 are the same, then the 3: LOAD instruction reads from a location that the *2: STORE* instruction modifies. In other words, the *3: LOAD* instruction is data dependant upon the *2: STORE* instruction.

- On the other hand, if the two addresses are different, then the two instructions are not dependent.

The uncertainty whether contents of R1 and address 300 refer to the same address cannot be resolved until the *2: STORE* and *3: LOAD* instructions have at least partially executed because at that point the processor can ascertain the content of R1. Because such data dependences may or may not exist, they are referred to as *ambiguous* data dependences. *See* Umberger Decl. Ex. 1, Col. 2, ll. 7-12.

V. Modern Processors Speculatively Execute LOAD Instructions That Have Ambiguous Data Dependences[7]

In the presence of an ambiguous dependence between two instructions, a processor may take a conservative view and assume that the two instructions are in fact dependent and execute them in order. *See* Umberger Decl. Ex. 1, Col. 2, ll. 21-24. In other words, the dotted arrow (ambiguous dependence) is treated as a solid arrow (unambiguous dependence). In the above sequence, the instructions will be executed in the order shown in Figure 4(a). When the processor encounters the *3: LOAD* instruction, it will delay the instruction until it can confirm that the execution of the instruction will not produce an incorrect result.

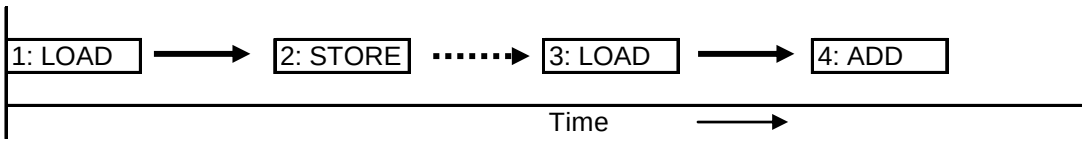

Fig. 4(a) – Sequential Execution of Instructions Because of Data Dependency

Figure 4(b) illustrates the sequential execution of these four instruction in terms of the four stages of execution: *fetch* (F), *issue* (I), *execute* (X) and *commit* (C). Because of data

[7] *See* Dally Decl. ¶¶ 31-40.

dependencies, each instruction is executed only after the preceding instruction has executed. For example, *2: STORE* is executed in cycle 13 after *1: LOAD* completes execution in cycle 12; and *3: LOAD* executes after *2: STORE*. The instructions commit (C) in the order in which they appear in the program.

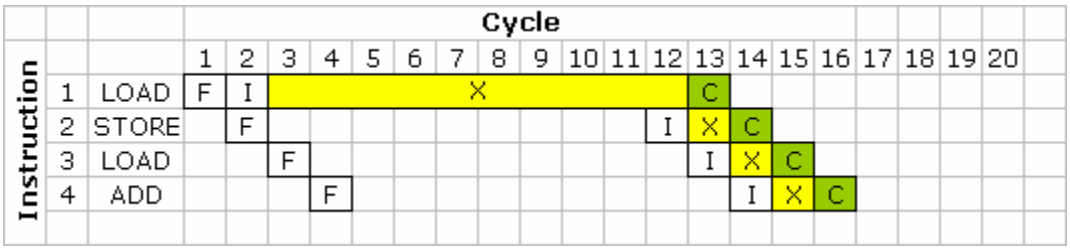

Fig. 4(b) – Detailed view of Sequential Execution of Instructions

However, it is possible that *3: LOAD* instruction is not data dependent upon the *2: STORE* instruction. In that event, the *3: LOAD* and *4: ADD* instructions could have at least partially overlapped and executed in parallel with the *1: LOAD* and *2: STORE* instructions, as shown below in Figure 5(a):

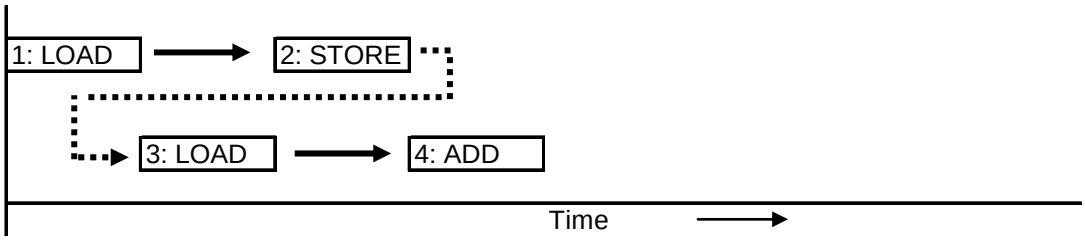

Fig. 5(a) – Parallel Execution of 3: LOAD and 4: ADD Instructions

Comparison of Figures 4(a) and 5(a) demonstrates that by treating ambiguous data dependences as actual data dependences, the processor is not able to extract all parallelism in a program (*i.e.*, it is not as efficient as it could be).

Modern processors attempt to capture this potential parallelism lost due to ambiguous dependences by executing LOAD instructions in a data speculative manner. *See* Umberger Decl. Ex. 1, Col. 2, ll. 26-30. A processor capable of data speculation allows execution of instructions

even before the processor can determine whether the execution of the instruction will produce a correct result. *See* Umberger Decl. Ex. 1, Col. 2, ll. 36-40. In other words, the processor *speculates* (or makes an assumption) that the LOAD instruction is not data dependent on an earlier instruction and proceeds to execute the LOAD instruction as if it were not data dependent. However, the processor keeps track of such speculatively executed LOAD instructions and subsequent instructions that are dependent on those LOAD instructions in the event the speculation was incorrect. At a later stage, the processor verifies the correctness of its speculation (*i.e.*, assumption) that the LOAD was not data dependent. If the speculation was correct, the speculatively executed instructions are deemed to have executed correctly. If the speculation was incorrect, a mis-speculation has occurred, *i.e.*, a LOAD that depends on a STORE for its data has in fact executed before the STORE. The speculatively executed instructions are canceled or *squashed* and re-executed. *See* Umberger Decl. Ex. 1, Col. 2, ll. 42-56.

For illustration, consider how the four instruction sequence may be executed in a data speculative manner:

```
1:   LOAD Memory[200], R1
2:   STORE R2, Memory[R1]
3:   LOAD Memory[300], R3
4:   ADD R3, #16, R4
```

If a processor allows data speculation, then the processor will speculate (*i.e.*, assume) that the *3: LOAD* instruction is not data dependent. That is, the processor will not wait for the *2: STORE* instruction to finish before issuing the *3: LOAD* instruction. Figure 5(b) provides a detailed view of speculative execution of these four instructions that captures the parallelism shown in Figure 5(a) above:

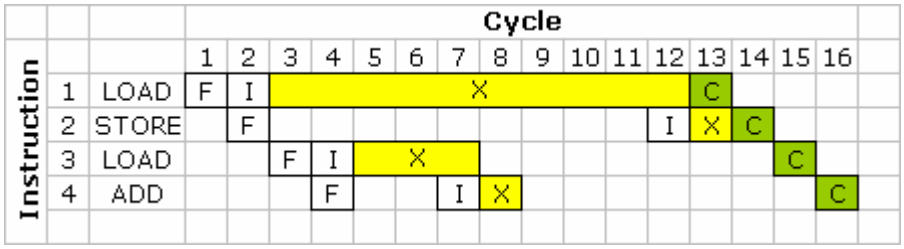

Fig. 5(b) – Speculative Execution of 3: LOAD and 4: ADD Instruction

When the processor encounters the *3: LOAD* instruction, it does not delay the LOAD instruction until the *2: STORE* instruction has executed. Instead, the *3: LOAD* is executed speculatively in cycles 5 through 7. The *2: STORE* does not execute until cycle 13. Even though the *3: LOAD* instruction depends upon the *2: STORE* instruction, the *3: LOAD* is executed before the *2: STORE*. The *4: ADD* instruction is also issued speculatively in cycle 7 because it depends upon the *3: LOAD* instruction. Thus, as shown in Figure 5(a), the execution of instructions *3: LOAD* and *4: ADD* overlaps with the execution of instructions *1: LOAD* and *2: STORE*. Note that though instructions finish execution in different order, they still commit (C) sequentially.

In the event that the *2: STORE* and *3: LOAD* instruction were not data dependent (*i.e.*, the processor's speculation was correct), the instructions *3: LOAD* and *4: ADD* may be executed before instructions *1: LOAD* and *2: STORE*. The program order (order in which instructions occur in a computer program) and the execution order (the order in which the instructions are executed) of these instructions is shown in Figure 6 below. Thus, the instructions 3 and 4 that occur after instructions 1 and 2 in program order are executed earlier than instructions 1 and 2. This is an example of *out of order* (non-sequential) execution. It noteworthy that even in the *out of order* execution sequence, the *unambiguous* data dependences have been observed. However, the *3: LOAD* may be executed before the *2: STORE* instruction because the ambiguous data dependence between the two instructions did not in fact exist.

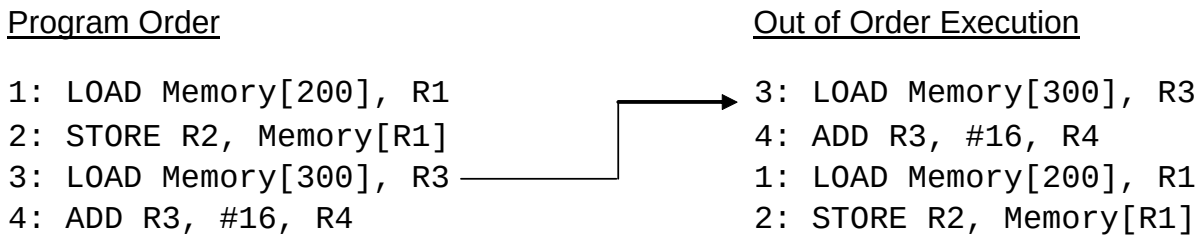

Fig. 6 – Example of an Out of Order Execution Sequence

On the other hand, if the *3: LOAD* instruction did depend on the *2: STORE* instruction (*i.e.*, the speculation was incorrect), a mis-speculation is declared. The *3: LOAD* instruction and the *4: ADD* instruction have relied on incorrect data and must be squashed and re-executed. Figure 6 illustrates how this is handled. Prior to committing its results in cycle 14, the *2: STORE* instruction checks whether any LOAD instruction has accessed or is bound to access data updated by the STORE instruction. In this example, the *3: LOAD* instruction finished execution in cycle 7 and thus, fetched incorrect data. The processor detects the mis-speculation and squashes the *3: LOAD* instruction in cycle 14. It also squashes the *4: ADD* because the ADD relied on incorrect data fetched by the *3: LOAD*. The two squashed instructions are subsequently re-executed in cycles 16 and 17.

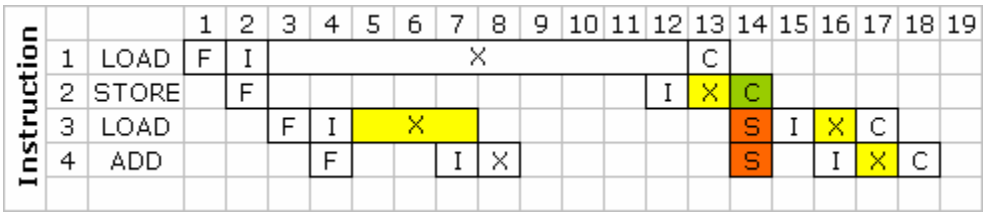

Fig. 7 – Example of a Mis-speculation, *i.e.*, Incorrect Speculation

Squashing speculatively executed instructions and re-executing them incurs performance penalties. *See* Dally Decl. ¶¶ 31-32; Umberger Decl. Ex. 1, Col. 2, ll. 51-56; Col. 3, ll. 12-20. If mis-speculation is rare, then the benefits of data speculative execution outweigh the costs of mis-speculation and the performance of the processor is improved by enabling data speculation.

However, if mis-speculation occurs often, then the costs of mis-speculation can easily exceed the benefits of data speculation.

VI. Modern Processors Employ A Memory Hierarchy To Improve Performance Of LOAD And STORE Instructions[8]

Modern high-performance processors implement their data memory as multiple levels of memory as shown in Figure 8 below. These levels of memory are referred to as the "memory hierarchy." The levels of hierarchy closest to the processor are fast but small. As one moves to the right in the figure, the capacity of the memory levels gets larger but the memory level gets slower.

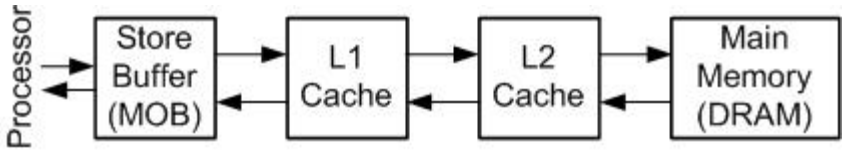

Fig. 8 – Memory Hierarchy

During execution, a LOAD instruction will first access the Store Buffer in the MOB to locate the data. If the data is found, then the LOAD instruction is done. If not, the LOAD instruction will access the L1 cache. If the data is not found in the L1 cache, the LOAD instruction will access other levels of the memory hierarchy until the data is found. A consequence of the memory hierarchy is that it is hard to determine *a priori* how long a particular LOAD operation will take to execute. A LOAD instruction that has to access several levels of memory hierarchy takes longer to retrieve data than a LOAD instruction that finds the data in the Store Buffer of the MOB. An example is shown below in Figure 9:

[8] *See* Dally Decl. ¶¶ 24-30.

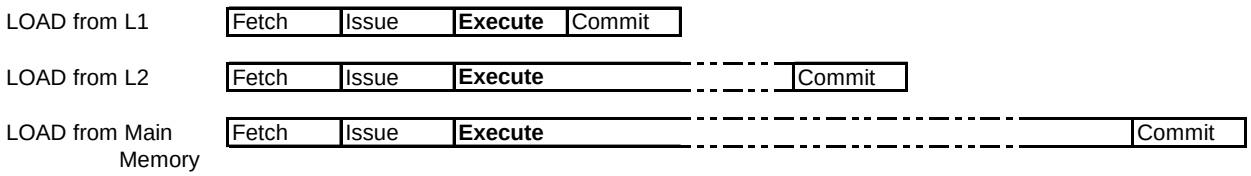

Fig. 9 – LOAD Instructions Accessing Different Levels of Memory Hierarchy

VII. The '752 Patent Introduced The Concept Of Selective Data Speculation Of Load Instructions[9]

If a processor does not perform data speculation on LOAD instructions, then as discussed in Section V above, that processor is not able to achieve optimal performance. On the other hand, if the processor executes all LOAD instructions speculatively, then it incurs a performance penalty whenever a mis-speculation occurs. If a processor mis-speculates often, then the cost of mis-speculation can easily outweigh any performance gains attributable to data speculation. The prior art processors could not decide when to data speculate on an instruction basis – either all LOAD instructions were speculatively executed or none at all. Umberger Decl. Ex. 1, Col. 3, ll. 8-11; *see also* Dally Decl. ¶¶ 41-43.

As a result, attempts were made to identify schemes between the two extremes that would permit control of data speculation at a finer granularity. For example, the scheme proposed in U.S. Patent No. 5,666,506 (cited during the prosecution of the '752 patent), inhibits data speculation when a STORE instruction that has mis-speculated in the past is being processed. *See* Umberger Decl. Ex. 3. As a result, this scheme indiscriminately delays **all** LOAD instructions while the offending STORE is being handled. *See* Dally Decl. ¶ 42. This proposal is flawed because by blocking all LOAD instructions, it unnecessarily blocks LOAD instructions that do not conflict with the offending STORE, and thus, loses performance. *See id.*

[9] *See* Dally Decl. ¶¶ 44-48.

In another scheme proposed in U.S. Patent No. 5,619,662, when a mis-speculation is detected between a LOAD and a STORE instruction, the pair is tagged and never reordered. *See* Umberger Decl. Ex. 7. Thus, the pair is indiscriminately predicted to always mis-speculate even though the pair may never conflict during the remainder of the program execution. *See* Dally Decl. ¶ 43. By failing to adapt to the program behavior, this scheme is unable to capture all potential performance gains. *See id.*

A. The '752 Patent Teaches Improved Selective Speculation Of LOAD Instructions

The '752 invention is a significant improvement over the prior art schemes because it is a refined mechanism that allows a processor to ***accurately select*** LOAD instructions that should be executed in a data speculative manner.[10] *See* Dally Decl. ¶¶ 44-45. Improvement in selectivity leads to improved performance by the processor. *See* Dally Decl. ¶¶ 44. The '752 invention associates a prediction value with LOAD instructions that tend to mis-speculate. *See* Umberger Decl. Ex. 1, Claim 1, Col. 10, ll. 7-11; Dally Decl. ¶ 47. The prediction indicates whether the speculative execution of that LOAD instruction is likely to result in another mis-speculation. The higher the prediction, the higher the likelihood that the LOAD instruction will cause a mis-speculation. A lower value of prediction corresponds to a lower likelihood of mis-speculation. *See* Dally Decl. ¶¶ 47; Umberger Decl. Ex. 1, Col. 11, ll. 31-33.

A processor embodying the invention of the '752 patent is able to leverage the benefits from data speculative execution while minimizing the costs incurred due to mis-speculation. Such a processor uses the prediction associated with the LOAD instruction to decide whether to execute the LOAD in a data speculative manner. Only those LOAD instructions that have a low

[10] LOAD instructions usually are referred to in the patent as "data consuming instructions"; STORE instructions are referred to as "data producing instructions."

likelihood of mis-speculation are speculatively executed. The LOAD instructions that are likely to cause mis-speculation are delayed until they can be executed in a non-speculative manner. Thus, instructions that are likely to result in a performance penalty through mis-speculation are not allowed to execute speculatively and the instructions that provide performance gains are allowed to proceed.

B. The '752 Patent Leverages Past Behavior Of LOAD Instructions To Predict Future Behavior

The '752 invention is able to intelligently decide when to data speculate on a LOAD instruction because its prediction mechanism is rooted in a phenomenon that is commonly exhibited by computer programs and was recognized by the inventors – if a LOAD instruction conflicts with a STORE instruction, then that LOAD instruction is highly likely to conflict with the same STORE instruction again. Simply put, the past behavior of a LOAD instruction is a good predictor of its future behavior.

The '752 invention uses the prediction values to capture the historical mis-speculation behavior of LOAD instructions. It tracks the execution of every LOAD instruction and updates the associated prediction values based on the outcome of those executions. For example, if a speculatively executed LOAD instruction mis-speculated, then the associated prediction value is incremented. On the other hand, if the LOAD did not mis-speculate, the prediction value is decremented. Because the past behavior of a LOAD instruction is a good indicator of its future behavior, a prediction informed by the past behavior of the LOAD instruction is able to identify those LOAD instructions that are likely to cause performance loss through mis-speculation.

C. The '752 Patent Describes Three Tiers Of Invention

The entire '752 invention can be implemented in a three-tiered manner, which reflects three decisions involving speculative execution of LOAD instructions. Umberger Decl. Ex. 1,

Col. 3, ll. 63-64; Dally Decl. ¶¶ 46-47. In the **first tier**, the processor checks whether the LOAD has mis-speculated in the past. If the LOAD instruction does not have a history of mis-speculation, the processor executes that instruction speculatively with the expectation that the speculation will be correct. Specifically, the patent provides:

> If there is no history of data mis-speculation, an instruction is executed without further inquiry. This will be the case for most data independent instructions.

Umberger Decl. Ex.1, Col. 3, ll. 64-67.

If the LOAD instruction has mis-speculated in the past, then the **second tier** is implicated. Umberger Decl. Ex.1, Col. 3, l. 67-Col. 4, l. 5. In this tier, the processor checks the prediction associated with that LOAD instruction. *Id.* Based on the value of the prediction, the processor decides either to speculatively execute the instruction or to delay the instruction. Regarding this tier, the patent states:

> If there has been a mis-speculation with a given LOAD instruction, a predictor based on the past history of mis-speculations for that LOAD instruction is employed to determine whether the instruction should be executed or delayed. Thus, instructions that are typically not dependent may be executed immediately.

Umberger Decl. Ex. 1, Col. 3, l. 67 – Col. 4, l. 5.

If the LOAD instruction is delayed, then the **third tier** is employed to decide *when* the delayed LOAD instruction should be executed. Umberger Decl. Ex. 1, Col. 4, ll. 5-7. The delayed LOAD instruction can be safely executed when the STORE instruction giving rise to the data dependence has been executed. To ascertain when the dependence has disappeared, the '752 invention maintains additional information:

- it saves the identity of the STORE instruction in the prediction table; and
- it maintains a synchronization table containing the identities of the LOAD and the STORE instruction pairs.

The additional hardware allows the processor to discern when the STORE instruction has executed and then to release the delayed LOAD instruction. It also allows the STORE instruction to forward the data directly from the STORE to the LOAD instruction.

The first-second and the third tiers of the invention reflect a performance vs. complexity tradeoff. The third tier, at the cost of additional complexity, allows the processor to precisely determine when the delayed LOAD instruction should be executed. A processor implementing only the first and second tiers provides coarser control but requires less hardware – it is be able to delay the appropriate LOAD instructions but cannot identify the time when the STORE causing the data dependence has executed. In such a processor, the delayed LOAD instruction would be released after all the pending STORE instructions have executed. The '752 patent claims both the less complex first-second tier invention (in an independent claim) as well as the higher performance/increased complexity third-tier invention (in dependent claims).

CONSTRUCTION OF DISPUTED CLAIM TERMS

The only asserted independent claim of the '752 patent, claim 1, recites as follows (WARF's proposed terms for construction in bold and underlined)[11]:

> 1. In a processor capable of executing program instructions in an execution order differing from their program order, the processor further having a data speculation circuit for detecting data dependence between instructions and detecting a mis-speculation where a [LOAD instruction] dependent for its data on a [STORE instruction] of earlier program order, is **in fact executed** before the [STORE instruction], a data speculation decision circuit comprising:
>
> a) a predictor receiving a mis-speculation indication from the data speculation circuit to produce a **prediction** associated with the [LOAD instruction] and based on the mis-speculation indication; and

[11] For ease of reading, WARF has replaced the term "data consuming instruction" (the nomenclature used in the '752 patent) with "LOAD instruction" and the term "data producing instruction" with "STORE instruction." There is no dispute as to the equivalence of these terms.

b) a prediction threshold detector preventing data speculation for instructions having a prediction within a predetermined range.

Umberger Decl. Ex. 1, Claim 1.

I. CONSTRUCTION OF THE DISPUTED TERM: "PREDICTION"

Below are the parties' proposed constructions of the term "prediction" as it appears in claim 1.

Claim Term	WARF's Proposal	Intel's Proposal
prediction associated with the particular data consuming instruction and based on the mis-speculation indication	a dynamic multi-bit value which indicates the likelihood that the data speculative execution of a LOAD instruction will result in a mis-speculation	a value indicating the likelihood that data speculative execution of a load/store pair will result in a mis-speculation

A. Why The Term Should Be Construed

Construction of this term is likely to be relevant to the validity of the '752 patent. The proper construction of the term "prediction" as used in the specification – which, as discussed below, unambiguously discloses that the claimed "prediction" is based on the LOAD instructions and is both dynamic and multi-bit – will assist in distinguishing the '752 patent from prior art (i.e., the applicability of certain prior art likely will be impacted by the scope of the Court's construction of this term). Further, this term is a term of art that carries a special meaning in light of the patent specification that would not be understood by the lay person (i.e., the factfinder) without further clarification. *See* Dally Decl. ¶ 62.

B. The Nature Of The Dispute

The key differences between WARF and Intel's proposed constructions are twofold:

- whether "prediction" is associated with a particular LOAD instruction or a particular LOAD/STORE instruction pair; and

- whether "prediction" as used in the '752 patent is "dynamic" and "multi-bit."

As discussed in detail below, prediction in the '752 patent predicts the likelihood that speculative execution of a particular LOAD instruction will result in mis-speculation, which mirrors WARF's proposal, not that speculative execution of a LOAD/STORE pair will result in mis-speculation. Further, the term "prediction" must be construed to reflect its dynamic aspect because the prediction value described in the '752 patent is continuously updated to track the varying data dependent behavior of the associated LOAD instruction as the program executes. And "prediction" must be construed to reflect that the value is multi-bit because it indicates the degree of likelihood that a LOAD instruction will mis-speculate and not the all-or-nothing scenario that a one-bit value would capture (which is not a "prediction" at all). Thus, these adjectives are essential to give the term "prediction" its appropriate meaning within the context of the '752 patent. For these reasons, WARF's proposed definition should be adopted.

C. Prediction Is Associated With A Particular LOAD Instruction And Not A LOAD/STORE Instruction Pair

1. Claim Language Associates Prediction With The LOAD Instruction

It is axiomatic that it is *the claims* of a patent that define the invention. *Phillips*, 415 F.3d at 1312 ("It is a bedrock principle of patent law that the claims of a patent define the invention to which the patentee is entitled the right to exclude.") (internal quotation marks omitted). The first limitation of claim 1 of the '752 patent provides:

> a predictor receiving a mis-speculation indication from the data speculation circuit to produce a **prediction associated with the particular data consuming instruction** [**i.e., LOAD**] and based on the mis-speculation indication;

Umberger Decl. Ex. 1, Claim 1, Limitation (a) (emphasis added). The language of claim 1 explicitly refers to a prediction associated with a LOAD instruction, not with a LOAD/STORE pair. Similarly, the inventors consistently associated prediction with only a LOAD instruction in

all other independent and dependent claims of the '752 patent. And, tellingly, the inventors knew how to refer to LOAD/STORE pairs (as indicated by the fact that they did so in other claims), but here they just refer to a LOAD. *Compare id. with, e.g.,* Umberger Decl. Ex. 1, Claim 5 ("the instruction synchronization circuit includes a synchronization table associating the certain data consuming instructions [LOADS] and the certain data producing instructions [STORES] . . . .") and claim 9 ("an entry listing a particular data consuming instruction [LOAD] and data producing instruction [STORE] each associated with a prediction . . . ."; *compare also* Col. 11, ll. 43-45 and Col. 14, ll. 15-18 (specification specifically referring to "LOAD/STORE pair(s)").

The reason the inventors chose to associate prediction with a LOAD instruction and not a LOAD/STORE pair is simple – data speculative execution applies only to LOAD instructions; STORE instructions are never speculatively executed. Dally Decl. ¶ 57. Hence, it does not make sense to refer to speculative execution of a LOAD/STORE pair. *Id.* It is the LOAD instruction that is speculatively executed, not the pair. In light of the disclosure in the '752 patent, a person of ordinary skill would understand that prediction in this patent provides the likelihood that data speculative execution of particular LOAD instruction will result in mis-speculation, not that data speculative execution of a LOAD/STORE pair will result in mis-speculation. *See* Dally Decl. ¶¶ 58-61.

Because Intel's proposed construction is contrary both to the plain language of the claims and to the common understanding of how LOAD and STORE instructions are executed in a processor capable of data speculation, it should not be adopted by this Court.

2. The '752 Patent Decides Whether To Speculate Or Not Based On Only The LOAD Instruction

As mentioned above, it does not make sense to refer to speculative execution of LOAD/STORE pairs because the STORE instructions are never executed speculatively. Therefore, in the '752 patent, the decision whether to speculatively execute a LOAD instruction or not implicates the LOAD instruction and its associated prediction. It does not involve the STORE instruction of a LOAD/STORE pair. Figure 3 and Figure 4 of the '752 patent describe the data speculation circuit and the predictor circuit respectively in one embodiment of the invention. Review of the communication between the blocks reveals that the predictor circuit makes a prediction as to whether the LOAD should be executed or delayed, and not regarding a LOAD/STORE pair. For example, in block 48 of Figure 3, the data speculation circuit checks whether it is handling a LOAD or a STORE instruction. If it is handling a LOAD instruction, then in block 66, the circuit checks whether this LOAD instruction is data speculative or not. The specification describes this step as follows:

> If at decision block 48 the instruction received by the data speculation circuit 30 is a LOAD instruction, then at decision block 66 it is determined **whether this is a data speculative LOAD, that is whether there are prior STORE instructions on which it might depend**.

Umberger Decl. Ex. 1, Col. 10, ll. 7-11 (emphasis added). *See also* Dally Decl. ¶ 58. The '752 invention specifically checks whether the LOAD instruction has a history of mis-speculation, and **not**, whether the LOAD/STORE pair is data speculative. If the inventors had conceived their invention as limited to speculatively executing LOAD/STORE pairs, they would have referred to a STORE instruction above. Instead, the inventors understood that a LOAD instruction could depend on several STORE instructions.

The '752 invention then consults the predictor circuit to obtain the prediction associated with this LOAD instruction:

> The predictor circuit 33 will address the READY TO LOAD request from the data speculation circuit by making a prediction as to **whether the LOAD should take place through the use of a wait flag**.

Umberger Decl. Ex. 1, Col. 10, ll. 19-22 (emphasis added). In other words, the predictor provides a prediction whether the LOAD instruction can proceed or should be delayed. *See* Dally Decl. ¶ 58. Notably, the inventors did not state whether the data speculation for a LOAD/STORE pair can occur or not. Figure 4 describes the operation of the predictor circuit. In block 100 of that figure, the predictor checks whether the LOAD instruction is in the prediction table, and not a LOAD/STORE pair. *See* Dally Decl. ¶ 58.

Review of other decision blocks in Figure 3 further highlights that the inventors' intent to associate the prediction with a LOAD instruction and not with a LOAD/STORE pair. At process block 76 in Figure 3, a delayed LOAD instruction waits until it is squashed or until is no longer data speculative. The specification describes this block as follows:

> Next at process block 76, the data speculation circuit 30 waits for that particular instruction, either to be squashed indicating that it had been erroneously speculated or for an indication that it is no longer data speculative, **that is any previous STORE instructions were for different memory addresses**.

Umberger Decl. Ex. 1, Col. 10, ll. 25-34 (emphasis added). In other words, the LOAD instruction was delayed because it could have been dependent upon any of the previous STORE instructions. If all of the previous STORE instructions were for addresses different than the address for the LOAD instruction, then the LOAD instruction is no longer data speculative and can be executed. If the '752 invention was speculatively executing only on LOAD/STORE pairs, then the inventors would have mentioned that the LOAD was waiting for the particular STORE instruction from the LOAD/STORE pair.

The inventors' consistent reference to only the LOAD instruction and **not** the LOAD/STORE pair in the context of data speculative execution reveals the disparity between the correct nature of the '752 invention and Intel's proposed construction.

3. The Three-Tiered Approach Discussed In The Specification Is Mirrored In The Claims

As described above in Section VII.C, the inventors of the '752 patent envisioned that their invention could be implemented in a three-tiered manner which reflects three decision points associated with the execution of LOAD instructions: (1) whether the LOAD instruction is the prediction table; (2) if it is, whether the LOAD instruction should be delayed, i.e., whether data speculation should be prevents for the LOAD; and (3) if it is delayed, how long should the LOAD instruction be delayed. *See also* Dally Decl. ¶¶ 46-47. The preferred embodiment described in the patent implements all three of the tiers described above. *See* Dally Decl. ¶ 59. And the claims of the '752 patent have been drafted to mirror these tiers. *See* Dally Decl. ¶ 60.

It is not a coincidence that Claim 1 reflects the second tier. Claim 1 essentially involves obtaining the prediction for a LOAD instruction and then making a decision, based on that prediction, whether to prevent data speculation for that LOAD. The decision to delay a particular LOAD instruction requires only the prediction associated with that LOAD instruction and does not involve the corresponding STORE instructions.

The third tier involves deciding when to execute the LOAD instruction, which the inventors captured in the dependent claims of the '752 patent. Thus, the patent provides:

> If the instruction is to be delayed, a synchronization table is used to determine when the instruction is to be performed.

Umberger Decl. Ex. 1, Col. 4, ll. 5-7. *See also* Dally Decl. ¶¶ 46-47 (describing three tiers disclosed in the '752 patent). To support the third tier, which involves synchronizing and forwarding data between dependent LOAD/STORE pairs, the preferred embodiment includes an

identifier for a STORE instruction in the prediction table. Umberger Decl. Ex. 1, Fig 5, Col. 11, ll. 8-14. Also to facilitate implementation of the third tier, the '752 patent refers to the likelihood of dependence between a LOAD/STORE pair (Umberger Decl. Ex. 1, Col. 11, ll. 30-33, 43-45 and Col. 14, ll. 3-6) and updates the prediction value only for mis-speculations where the dependence is between a LOAD/STORE pair (Umberger Decl. Ex. 1, Col. 12, l. 65 – Col. 13, l. 3). The '752 patent observes that dependence between a small number of LOAD/STORE pairs are responsible for the majority of mis-speculations. Umberger Decl. Ex. 1, Col. 14, ll. 15-18. The inclusion of the STORE in the prediction table of the '752 patent and updating the prediction table when the mis-speculation is between a particular LOAD/STORE pair facilitate the third tier and are not required for prediction of mis-speculation. *See* Dally Decl. ¶¶ 59-60.

4. Intel Improperly Imports A Limitation From The Preferred Embodiment

Intel's proposed construction improperly conflates all three tiers into claim 1. As discussed above, the '752 invention maintains prediction on a LOAD-STORE pair basis in the prediction table only to enable the third tier of the invention. However, the preferred embodiment teaches that this is one good way to calculate the prediction for a LOAD instruction. As stated in the '752 patent, if a LOAD conflicts with a STORE, then that LOAD is highly likely to conflict with the same STORE again. In the preferred embodiment, a history of conflicts between a LOAD/STORE pair provides a very good estimate of how that LOAD would behave if executed speculatively. A person of skill ordinary skill would understand that the prediction can be calculated in several other ways depending upon which tiers of the invention are implemented. For example, if a processor embodies only tier 1 and tier 2, then the processor could maintain the predictions only for LOAD instructions without implicating the STORE instructions.

The key to understanding the meaning of "prediction" within the '752 patent again is the claim language itself, which places only two requirements on the prediction, namely that:

- the prediction be associated with a LOAD instruction; and
- the prediction be based on historical mis-speculation behavior of the LOAD instruction.

These two define how the prediction value is calculated. By seeking to limit the claimed prediction to one maintained for LOAD/STORE pairs only, Intel improperly imports how prediction is maintained in one embodiment into the claims, contrary to well-settled Federal Circuit precedent prohibiting such importation. *See, e.g., Superguide Corp. v. DirecTV Enters.*, 358 F.3d 870, 875 (Fed. Cir. 2004) ("[I]t is important not to import into a claim limitations that are not a part of the claim."); *Phillips*, 415 F.3d at 1323 ("[A]lthough the specification often describes very specific embodiments of the invention, we have repeatedly warned against confining the claims to those embodiments."); *id.* at 1314-15 (given its presence in the dependent claim, there is a "presumption that the limitation in question is not present in the independent claim") (citing *Laitram Corp. v. Rexnord, Inc.*, 939 F.2d 1533, 1538 (Fed. Cir. 1991); *Liebel-Flarsheim Co. v. Medrad, Inc.*, 358 F.3d 898, 910 (Fed. Cir. 2004)).

5. The Inclusion Of The Notion Of "Dynamic"

WARF's proposed construction captures the essence of the invention wherein the prediction is dynamically incremented or decremented based on the behavior of the LOAD instructions. *See, e.g.*, Umberger Decl. Ex. 1, Col. 4, ll. 1-3 ("a predictor based on the past history of mis-speculations for the LOAD instruction is employed"). As the specification makes amply clear, the "predictor circuit 33 provides a **dynamic indication** [i.e., a "prediction"] to the data speculation circuit 30 as to whether data speculation should be performed." Umberger Decl. Ex. 1, Col. 7, l. 67-Col. 8, l. 3 (emphasis added); *see also* Col. 4, ll. 31-33 ("Thus, it is one object

of the invention to provide a predictor circuit that may identify data dependences on an **on-going** or **dynamic** basis.") (emphasis added). *See also* Dally Decl. ¶ 56.

Intel's proposed construction, by contrast, ignores the dynamic, on-going aspect of "prediction" as used in the claims.

6. The Inclusion Of The Notion Of "Multi-Bit"

WARF's construction again remains true to what a person of ordinary skill in the art would understand the '752 patent to claim.[12] The patent explicitly calls for complex manners in which the "prediction" regarding two instructions can be maintained:

> It will be understood that the prediction 109 may be obtained by methods other than simply incrementing it in value for each speculation as is described herein. For example, various weighting schemes can be provided to cause the predictor circuit 33, for example, to be less sensitive to the earliest mis-speculations. More complex pattern matching techniques may be also used, for example, to catch situations where mis-speculations occur in groups or regular patterns.

Umberger Decl. Ex. 1, Col. 14, ll. 6-14. *See also, e.g.,* Umberger Decl. Ex. 1, Col. 4, ll. 4-5 ("[I]nstructions that are *typically* not dependent may be executed immediately.") (emphasis added).

Thus, the higher the prediction, the more the likelihood of mis-speculation; similarly, the lower the prediction, the less the likelihood of mis-speculation. Only a *multi-bit* value can capture a range of predictions and accommodate the scheme contemplated above. If this could be done with only one bit, then the "prediction" could take on only two values (a 0 and a 1), and would essentially constitute an on/off decision. This is not the invention claimed in the '752 patent; rather, for each LOAD instruction, the claimed invention maintains a multi-bit value. *See also* Dally Decl. ¶¶ 45 and 56.

[12] A description of a person of ordinary skill in the art at the time of the invention can be found in Professor Dally's Declaration. *See* Dally Decl. ¶¶ 49-52.

Indeed, Intel's own proposed construction is inconsistent with the inclusion of an on/off decision as a "prediction." In its proposed construction, Intel agrees (subject to the dispute regarding LOAD/STORE pairs discussed above) that the prediction "value" must reflect a "likelihood" of mis-speculation. Yet Intel continues to reject a construction that would recognize the multi-bit aspect of the prediction, instead being inclusive of a one bit or on/off decision. While a decision to speculate can be based on a single bit (as in the prior art), predictions (i.e., "likelihoods") are based on *several* historical mis-speculations. If you bring it down to one bit, the decision would always be based on the last speculation, which would limit the broader scope of the learning mechanism claimed in the '752 patent. A one-bit value, which essentially turns speculation on or off based on one data point, does not reflect a "likelihood" (i.e., a "prediction") at all. A person of ordinary skill in the art reading the '752 patent would understand as much.

II. CONSTRUCTION OF THE DISPUTED TERM: "IN FACT EXECUTED"

The parties' proposed constructions of "in fact executed" are set forth below:

Claim Term	WARF's Proposal	Intel's Proposal
in fact executed	a LOAD instruction is "in fact executed" before the STORE instruction when the LOAD instruction has actually accessed or was certain to access data that has not yet been updated by the STORE instruction	a load instruction is "in fact executed" when the load instruction actually has loaded data from a memory location

A. Why The Term Should Be Construed

A lay-person's definition of the term *in fact executed* does not capture the full scope of this term as it would be understood by one of ordinary skill in the art at the time of the invention. Nor does Intel's proposed construction of the phrase – that the LOAD instruction *actually has loaded data from a memory location* – capture the full scope of this term because it carves out scenarios that a skilled person would understand to be within the scope of the term. Because this

phrase is a term of art that carries a special meaning in light of the patent specification that would not be understood by a lay person (*i.e.*, the factfinder) without further clarification, construction by the Court is appropriate. *See* Dally Decl. ¶ 73.

B. How The Term Should Be Construed

1. The Claim Term Must Be Construed In Context

"In fact executed" is used in the preamble of claim 1:

> In a processor capable of executing program instructions in an execution order differing from their program order, the processor further having a data speculation circuit for detecting data dependence between instructions and detecting a mis-speculation where a [LOAD instruction] dependent for its data on a [STORE instruction] of earlier program order, is **in fact executed** before the [STORE instruction],

The claim makes it clear that the term *in fact executed* occurs in the context of detecting a mis-speculation, which occurs when a LOAD instruction dependent for its data on a STORE instruction is *in fact executed* before the STORE instruction.

As explained below, upon closer inspection of how data dependent LOAD and STORE instructions operate in an out-of-order processor, mis-speculation occurs when a LOAD instruction has either accessed incorrect data or is bound to access incorrect data before the STORE will update that data. In both scenarios, the LOAD instruction should be discarded because it had either accessed incorrect data or will do so. *See* Dally Decl. ¶ 68. As used in the patent, this term would be understood by those of skill in the art to mean that once the LOAD instruction has started the process of accessing data from the memory, it has "in fact executed" from the perspective of the STORE instruction. It is not necessary for the LOAD instruction to actually access the data to have "in fact executed." *See* Dally Decl. ¶¶ 67, 70 and 73.

By limiting the term to only those situations where the LOAD has "actually loaded data," Intel's proposed construction improperly ignores the context in which the phrase is used – upon

which context the meaning of the phrase is highly dependent. *See, e.g.*, *Phillips*, 415 F.3d at 1313 (a person of ordinary skill in the art "is deemed to read the claim term not only in the context of the particular claim in which the disputed term appears, but in the context of the entire patent, including the specification").

2. The Claim Term Must Be Given Its Full And Proper Scope

The heart of the dispute between WARF and Intel is whether the patent limits itself to only the LOAD instructions that have already accessed incorrect data, as Intel argues, or does it also include LOAD instructions that are being performed by processing units and are certain to access incorrect data, as WARF argues. WARF's construction is correct because the '752 patent never restricts itself to the LOAD instructions in the first scenario. *See* Dally Decl. ¶¶ 66-72. To the contrary, as shown below, the patent explicitly contemplates both situations. WARF's definition also reflects the understanding of those skilled in the art at the time the '752 patent was filed.

a. The '752 Patent Encompasses LOAD Instructions That Have Accessed Incorrect Data

The first situation noted above (LOAD instructions that have already accessed incorrect data) is mentioned in the '752 patent when discussing general processor architecture:

> Whenever a store instruction is ready to commit and write its data to a memory address, **the data speculation circuit 30, checks to see if any subsequent [load instructions] in the instruction window . . . have accessed the same memory address**, and if so instructs the allocation circuit 20 and retirement circuit 26 that these load instructions are to be squashed and re-allocated by the allocation circuit 20 at a later time.

Umberger Decl. Ex. 1, Col. 7, ll. 41-48 (emphasis added).[13] When the STORE instruction is about to commit, *i.e.*, about to makes its changes permanent and inform the rest of the system of those changes, the data speculation circuit checks whether any LOAD instructions have improperly accessed data that the STORE instruction updated. If that is the case, such LOAD instructions have to be squashed. Consider speculative execution of these four instructions:

```
1:   LOAD Memory[200], R1
2:   STORE R2, Memory[R1]
3:   LOAD Memory[300], R3
4:   ADD R3, #16, R4
```

An ambiguous dependence exists between the *3: LOAD* instruction and the *2: STORE* instruction. In this example, the *3: LOAD* instruction is speculatively issued and executed. This is illustrated in Figure 10[14] (which is same as Figure 7), where the *3: LOAD* is issued (I) in cycle 4 and finishes execution (X) in cycle 7 whereas *2: STORE* completes execution in cycle 13.

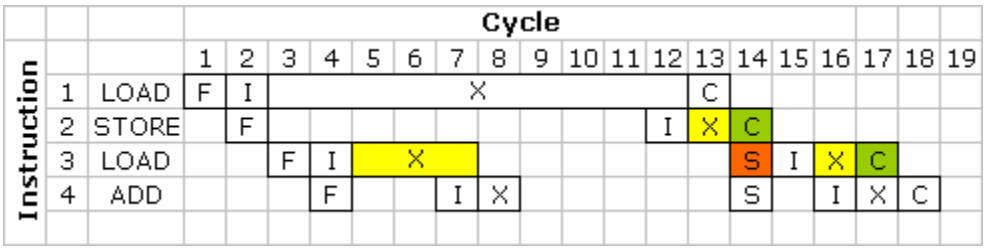

Fig. 10 – Example of a Mis-speculation when 3: LOAD has Accessed Incorrect Data

Prior to committing its results in cycle 14, the *2: STORE* instruction detects the mis-speculation by *3: LOAD* instruction. At that point, the speculatively issued *3: LOAD* instruction has already accessed the incorrect data when it completed its execution in cycle 7 (*i.e.*, before the

[13] This quotation from the patent has been altered (as indicated by the brackets and ellipses) to account for a drafting error in the placement of the term "load instructions," and such alterations do not change the meaning of the passage.

[14] Abbreviations denote: fetch (F); execute (X); issue (I); squash (S) and commit (C).

STORE instruction). The *3: LOAD* instruction is squashed in cycle 14 (denoted by "S") and subsequently re-executed. *See* Dally Decl. ¶¶ 34 and 68.

b. The '752 Patent Also Encompasses LOAD Instructions That Were Certain To Access Incorrect Data

While Intel's proposed construction would limit the claim scope to the scenario discussed above, the '752 patent also discloses LOAD instructions that are being performed by processing units and are being tracked in order to detected mis-speculations:

> The data speculation circuit 30 receives signals from the allocation circuit 20 that notify it of the program order of **any instructions that are allocated to the processing units 24 and that will access memory**. The data speculation circuit is responsible of keeping track of order of the memory operations **as they are performed by the processing units** so that it can detect any mis-speculations.

Umberger Decl. Ex. 1, Col. 7, ll. 1-7 (emphases added); *see also* Umberger Decl. Ex. 1, Col. 9, ll. 61-64) ("the data speculation circuit 30 checks other concurrent LOAD instructions to see if they have been prematurely executed and thus whether there has been a mis-speculation").

The data speculation circuit keeps track of memory operations (*i.e.*, LOAD and STORE instructions) as they are being performed so that the circuit can detect mis-speculations. Thus, it includes LOAD instructions that are being processed by the processing units (even if processing is not complete). And the data speculation circuit tracks in-progress LOAD instructions because in the event they violate a data dependence, they are certain to access incorrect data.

As discussed above in Section VI of the Background of the Technology, modern computer systems use memory hierarchies to improve performance of LOAD operations. *See also* Dally Decl. ¶¶ 24-30. A consequence of employing a memory hierarchy is that some LOAD instructions take longer to execute than other LOAD instructions. Thus, if a LOAD instruction takes a very long time, then that LOAD instruction may not have actually loaded data

from the memory location when a prior STORE is ready to commit. *See* Dally Decl. ¶ 69 and 72.

Returning to the above example, if the *3: LOAD* instruction takes a very long time, then the *3: LOAD* instruction would not have actually loaded data from the memory location when the *2: STORE* instruction is ready to commit in cycle 13. This scenario is depicted in the Figure 11 below, which shows that the *3: LOAD* instruction finished execution in cycle 97. Because the *3: LOAD* instruction depends upon the *2: STORE* instruction, the LOAD instruction, if allowed to execute unchecked, could fetch incorrect data. If the data speculation circuit is limited to checking only those LOAD instructions that have actually loaded data before STORE executes, as Intel proposes, such a system would not detect the mis-speculation, and would not squash this incorrectly executed LOAD instruction. *See* Dally Decl. ¶ 72.

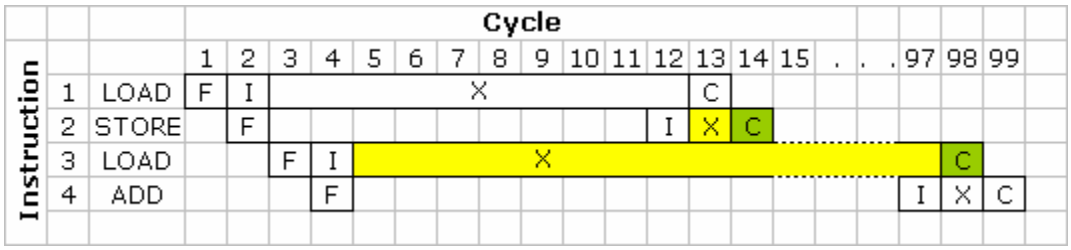

Fig. 11 – Example of when execution of 3: LOAD takes a long time

Thus, a LOAD instruction that has read incorrect data remains undetected and this will lead to incorrect program execution. Intel's proposed construction therefore cannot be correct. *See Cordis Corp. v. Medtronic Ave, Inc.*, 511 F.3d 1157, 1174 (Fed. Cir. 2008) ("a construction that renders the claimed invention inoperable should be viewed with extreme skepticism") (citations omitted).

On the other hand, a system operating according to WARF's definition (consistent with the specification as understood by one in the art) would capture and rectify this erroneous situation. In the above example, in cycle 13, when the STORE is about to commit, the data

speculation circuit knows the address that the LOAD instruction is reading. It also knows that when the LOAD instruction finishes, it may fetch incorrect data. Thus, it makes sense to squash that LOAD instruction right away to ensure that the program executes correctly. Thus, as shown in Figure 12 below, the *3: LOAD* instruction is squashed in cycle 14 even though it did not actually load data. *See* Dally Decl. ¶¶ 68, 69 and 72.

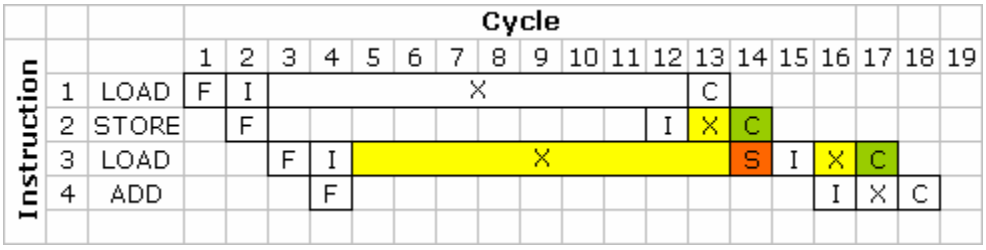

Fig. 12 – Example of Mis-speculation when 3: LOAD was certain to access Incorrect Data

3. WARF's Definition Reflects The Understanding Of The Person Of Ordinary Skill In The Art

A person of ordinary skill in the art, reading the claim language of '752 patent in context, would understand "in fact executed" to include both situations where the LOAD instruction *has actually accessed* and where the LOAD instruction *was certain to access data that has not yet been updated by the STORE instruction*. *See* Dally Decl. ¶¶ 66-69 and ¶ 70 (describing contemporaneously disclosed system (Umberger Decl. Ex. 5) wherein a LOAD that is blocked in this manner has *in fact executed* for one or more cycles before it is blocked).

CONCLUSION

For the foregoing reasons, Plaintiff respectfully requests that the Court construe the disputed claim terms proposed by WARF as follows:

The claim term "prediction" should be construed as "a dynamic multi-bit value which indicates the likelihood that the data speculative execution of a LOAD instruction will result in a mis-speculation"; and

The claim term "in fact executed" should be construed as "a LOAD instruction is 'in fact executed' before the STORE instruction when the LOAD instruction has actually accessed or was certain to access data that has not yet been updated by the STORE instruction."

Dated this 26th day of June, 2008.

HELLER EHRMAN LLP

By: */s/ Michelle M. Umberger*

John S. Skilton, SBN 1012794
John.Skilton@hellerehrman.com
Michelle M. Umberger, SBN 1023801
Michelle.Umberger@hellerehrman.com
Gabrielle E. Bina, SBN 1041749
Gabrielle.Bina@hellerehrman.com
Lissa R. Koop, SBN 1050597
Lissa.Koop@hellerehrman.com
One East Main Street, Suite 201
Madison, WI 53703-5118
Telephone: (608) 663-7460
Facsimile: (608) 663-7499

Robert T. Haslam
Robert.Haslam@hellerehrman.com
Anupam Sharma
Anupam.Sharma@hellerehrman.com
275 Middlefield Road
Menlo Park, CA 94025
Telephone: (650) 324-7000
Facsimile: (650) 324-0638

Attorneys for Wisconsin Alumni Research Foundation